C2SM
Center for Climate Systems Modeling

Technical Test Suite for COSMO

X. Lapillonne, N. Lardelli, O. Fuhrer

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

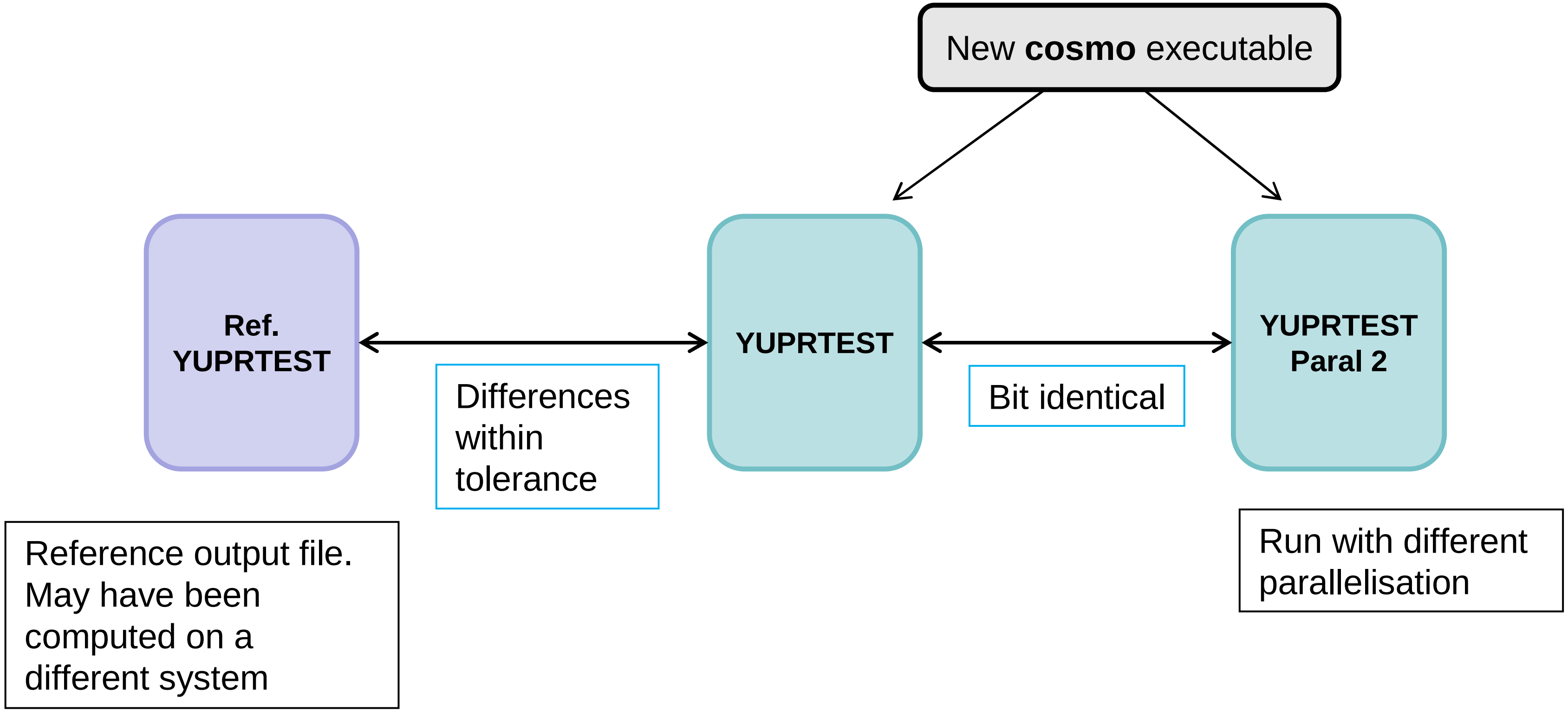
Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

Purpose of the technical Test Suite

- Light and easy to use python tool (testsuite.py) to check a newly developed COSMO model version:
  - The code is running and gives “correct” results with various configurations (e.g. only dynamics, dynamics + physics, members configurations …)
  - The code gives bit identical results with different processor configurations (including with or without I/O PE)
  - Restart functionality is working, and gives bit identical results
- Additional user defined verification could be specified
- Design to help addressing chapter 6.5 of COSMO standard: *Standard Test Suite*

Verifying Cosmo results

- ASCII output file (YUPRTEST) : double precision mean, max and min values at each vertical level of the prognostic fields
- Correct results: account for rounding error (i.e. which could arise from optimizations or use of a different compiler)
- Simulations time should be kept short (<1h)


New **cosmo** executable
Ref. YUPRTEST
YUPRTEST
YUPRTEST Paral 2
Differences within tolerance
Bit identical
Reference output file. May have been computed on a different system
Run with different parallelisation

Setting the tolerance factor

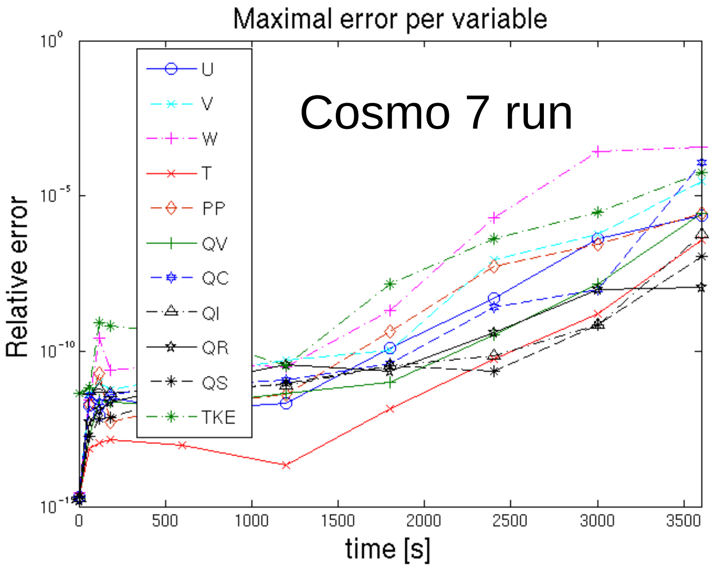
How to account for rounding error propagation ?

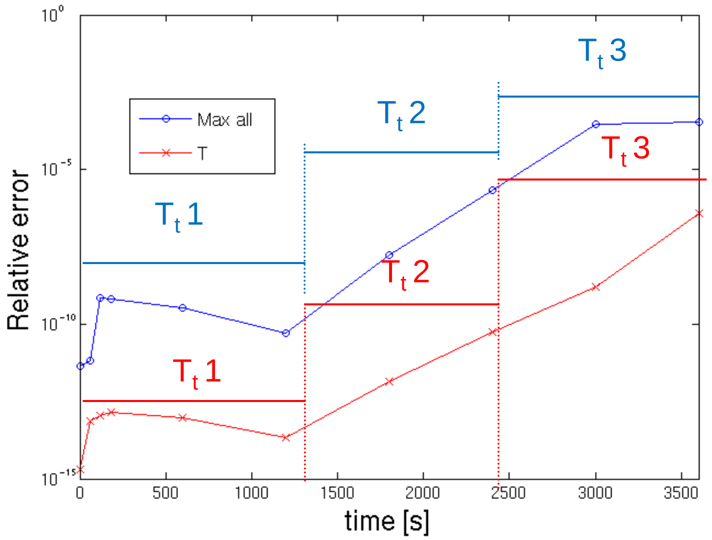
Methodology:

•2 perturbed cosmo executables compiled with different compilers

•At each step a perturbation is added to the prognostic fields:
f = f * (1+R*□) , with R random array and □ = 1^{-15}

•Run 30 experiments, compute maximal differences for each prognostic variables

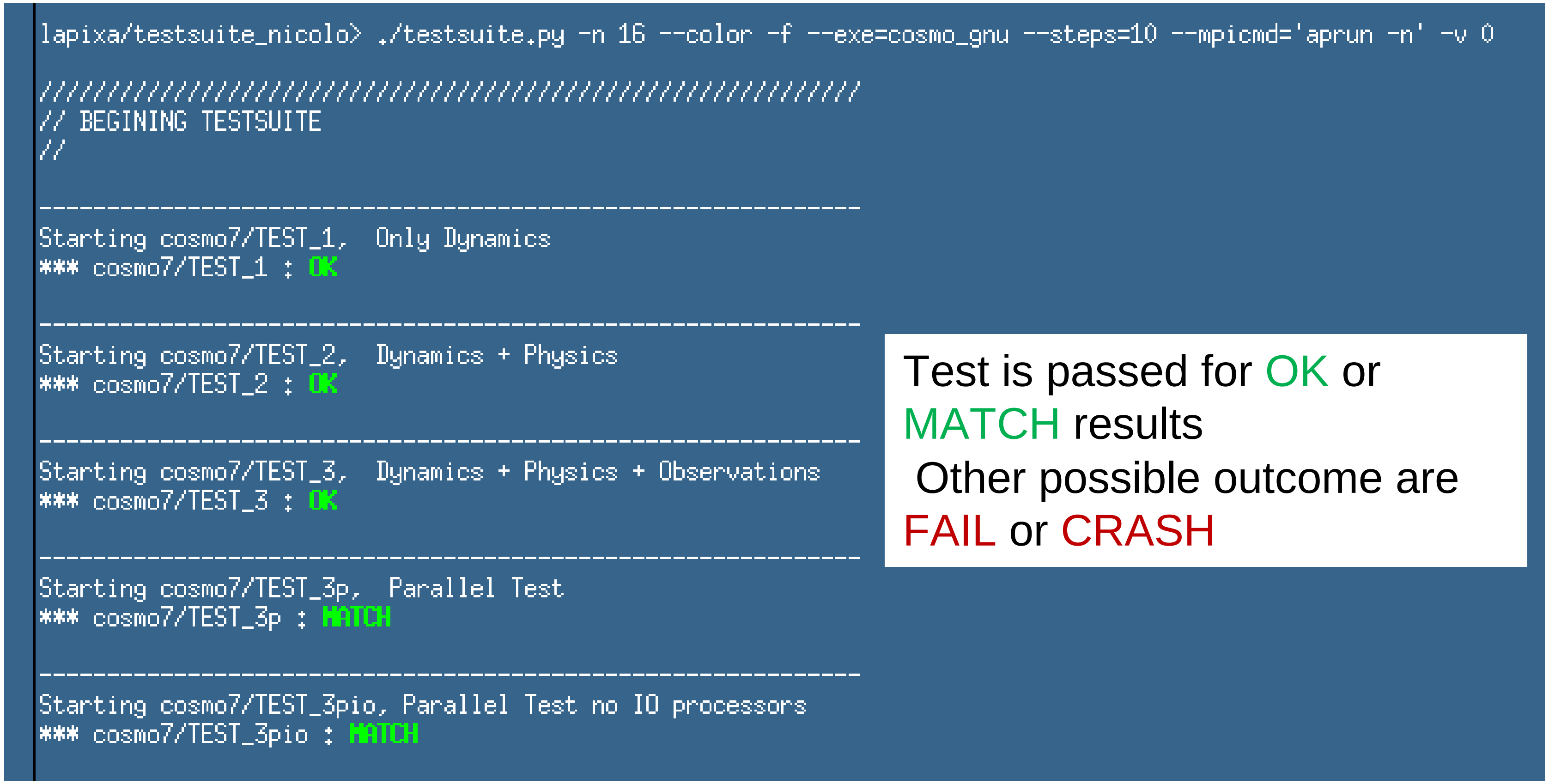




- Reduce tolerance number parameters : two groups of variables, T and All prognostics
- Set threshold for this 2 groups, for different time intervals
- Threshold can be set differently for different cases (e.g. cosmo2 or cosmo7)

Running testsuite.py

- The different tests are defined in an input file “testlist.xml”
- The script ./testsuite.py can be called with several command line arguments. (full description: ./testsuite.py –h)

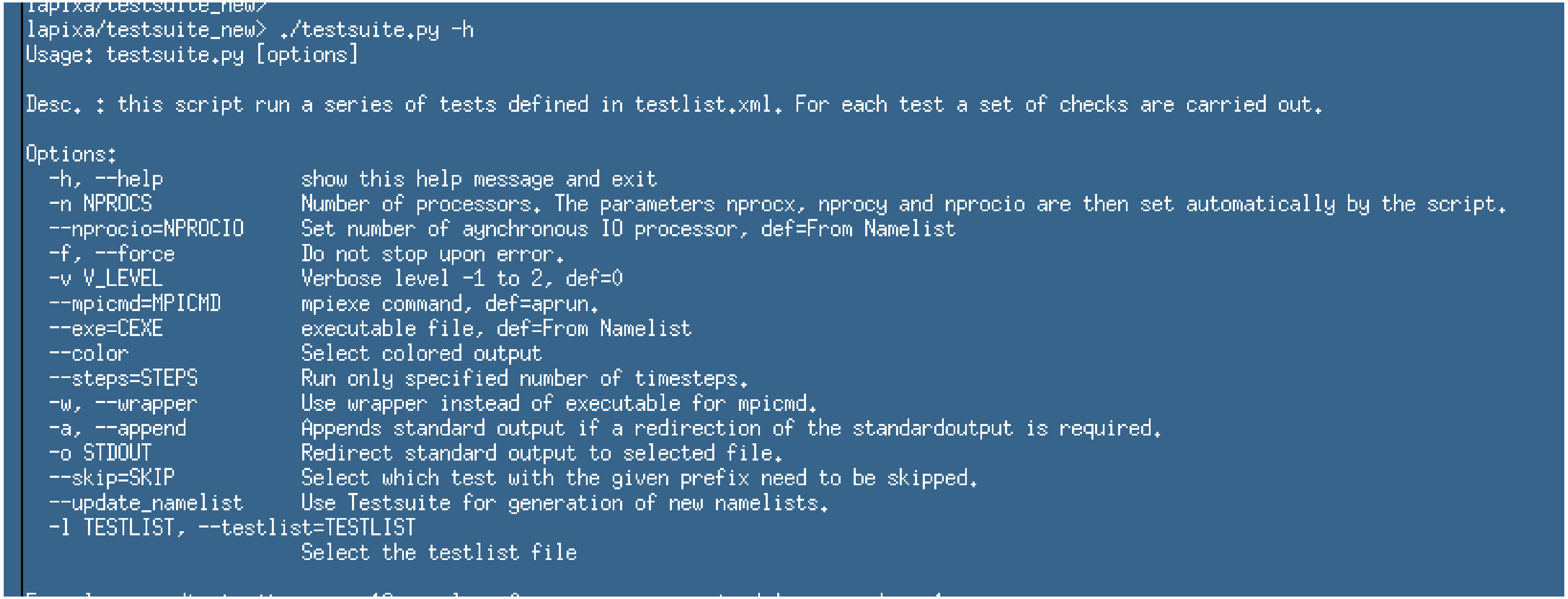


```
lapixa/testsuite_nicolo> ./testsuite.py -n 16 --color -f --exe=cosmo_gnu --steps=10 --mpicmd='aprun -n' -v 0

//////////////////////////////////////////////////////////////
// BEGINING TESTSUITE
//

--------------------------------------------------------------
Starting cosmo7/TEST_1,  Only Dynamics
*** cosmo7/TEST_1 : OK

--------------------------------------------------------------
Starting cosmo7/TEST_2,  Dynamics + Physics
*** cosmo7/TEST_2 : OK

--------------------------------------------------------------
Starting cosmo7/TEST_3,  Dynamics + Physics + Observations
*** cosmo7/TEST_3 : OK

--------------------------------------------------------------
Starting cosmo7/TEST_3p,  Parallel Test
*** cosmo7/TEST_3p : MATCH

--------------------------------------------------------------
Starting cosmo7/TEST_3pio, Parallel Test no IO processors
*** cosmo7/TEST_3pio : MATCH
```

Test is passed for OK or MATCH results

Other possible outcome are FAIL or CRASH

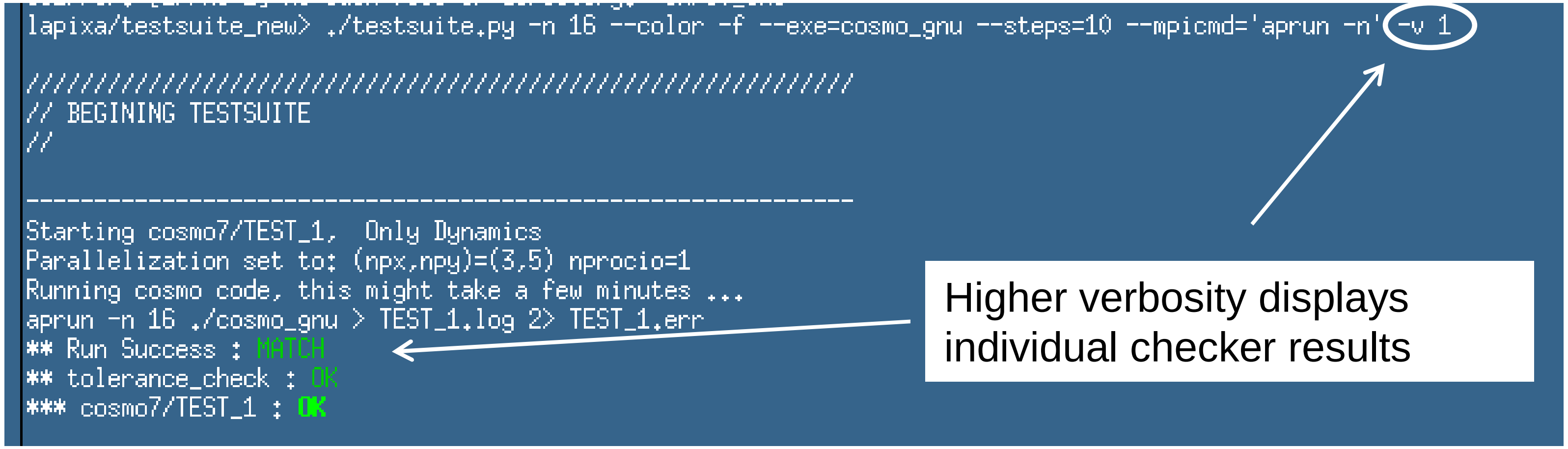
testsuite.py command line arguments



```
lapixa/testsuite_new>
lapixa/testsuite_new> ./testsuite.py -h
Usage: testsuite.py [options]

Desc. : this script run a series of tests defined in testlist.xml. For each test a set of checks are carried out.

Options:
  -h, --help            show this help message and exit
  -n NPROCS             Number of processors. The parameters nprocx, nprocy and nprocio are then set automatically by the script.
  --nprocio=NPROCIO     Set number of aynchronous IO processor, def=From Namelist
  -f, --force           Do not stop upon error.
  -v V_LEVEL            Verbose level -1 to 2, def=0
  --mpicmd=MPICMD       mpiexe command, def=aprun.
  --exe=CEXE            executable file, def=From Namelist
  --color               Select colored output
  --steps=STEPS         Run only specified number of timesteps.
  -w, --wrapper         Use wrapper instead of executable for mpicmd.
  -a, --append          Appends standard output if a redirection of the standardoutput is required.
  -o STDOUT             Redirect standard output to selected file.
  --skip=SKIP           Select which test with the given prefix need to be skipped.
  --update_namelist     Use Testsuite for generation of new namelists.
  -l TESTLIST, --testlist=TESTLIST
                        Select the testlist file
```

The checkers

- For each test a set of checkers can be called
- Checker : script (could be written in any language) that return one of the following exit code:
  0 : MATCH, 10 : OK , 20 : FAIL, 30 : CRASH
- Final test result is given by the max of individual checker results

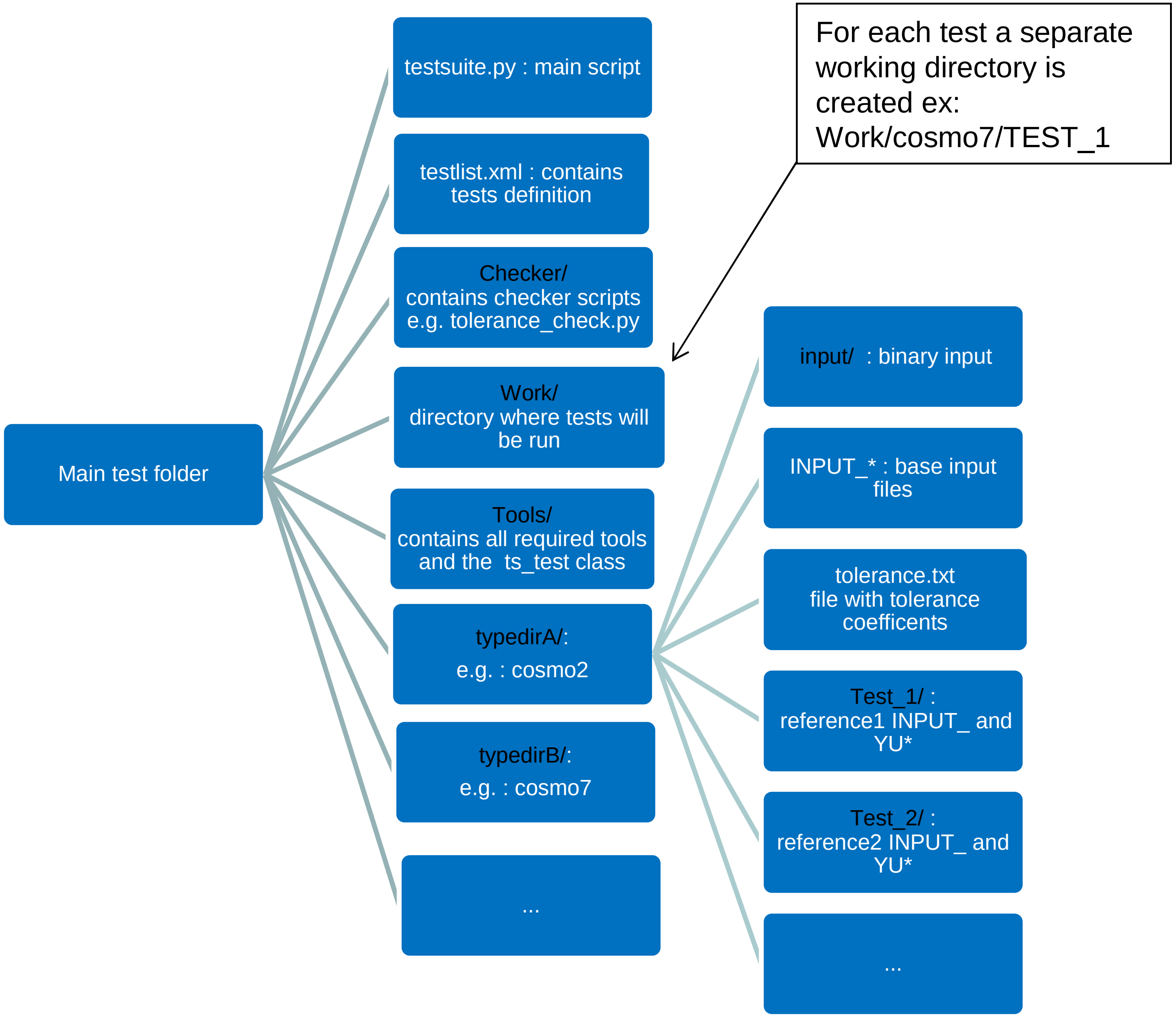


```
lapixa/testsuite_new> ./testsuite.py -n 16 --color -f --exe=cosmo_gnu --steps=10 --mpicmd='aprun -n' -v 1

//////////////////////////////////////////////////////////////////
// BEGINING TESTSUITE
//

------------------------------------------------------------------
Starting cosmo7/TEST_1,  Only Dynamics
Parallelization set to: (npx,npy)=(3,5) nprocio=1
Running cosmo code, this might take a few minutes ...
aprun -n 16 ./cosmo_gnu > TEST_1.log 2> TEST_1.err
** Run Success : MATCH
** tolerance_check : OK
*** cosmo7/TEST_1 : OK
```

Higher verbosity displays individual checker results

- The script can access run time environment variables (TS_BASEDIR, TS_NAMELISTDIR, TS_VERBOSE …) set by testsuite.py
- The idea is that each user can add his own custom checkers (ex: checking that a specific output file exists)

The testsuite directory



Main test folder

testsuite.py : main script

testlist.xml : contains tests definition

Checker/ contains checker scripts e.g. tolerance_check.py

Work/ directory where tests will be run

Tools/ contains all required tools and the ts_test class

typedirA/: e.g. : cosmo2

typedirB/: e.g. : cosmo7

...

input/ : binary input

INPUT_* : base input files

tolerance.txt file with tolerance coefficents

Test_1/ : reference1 INPUT_ and YU*

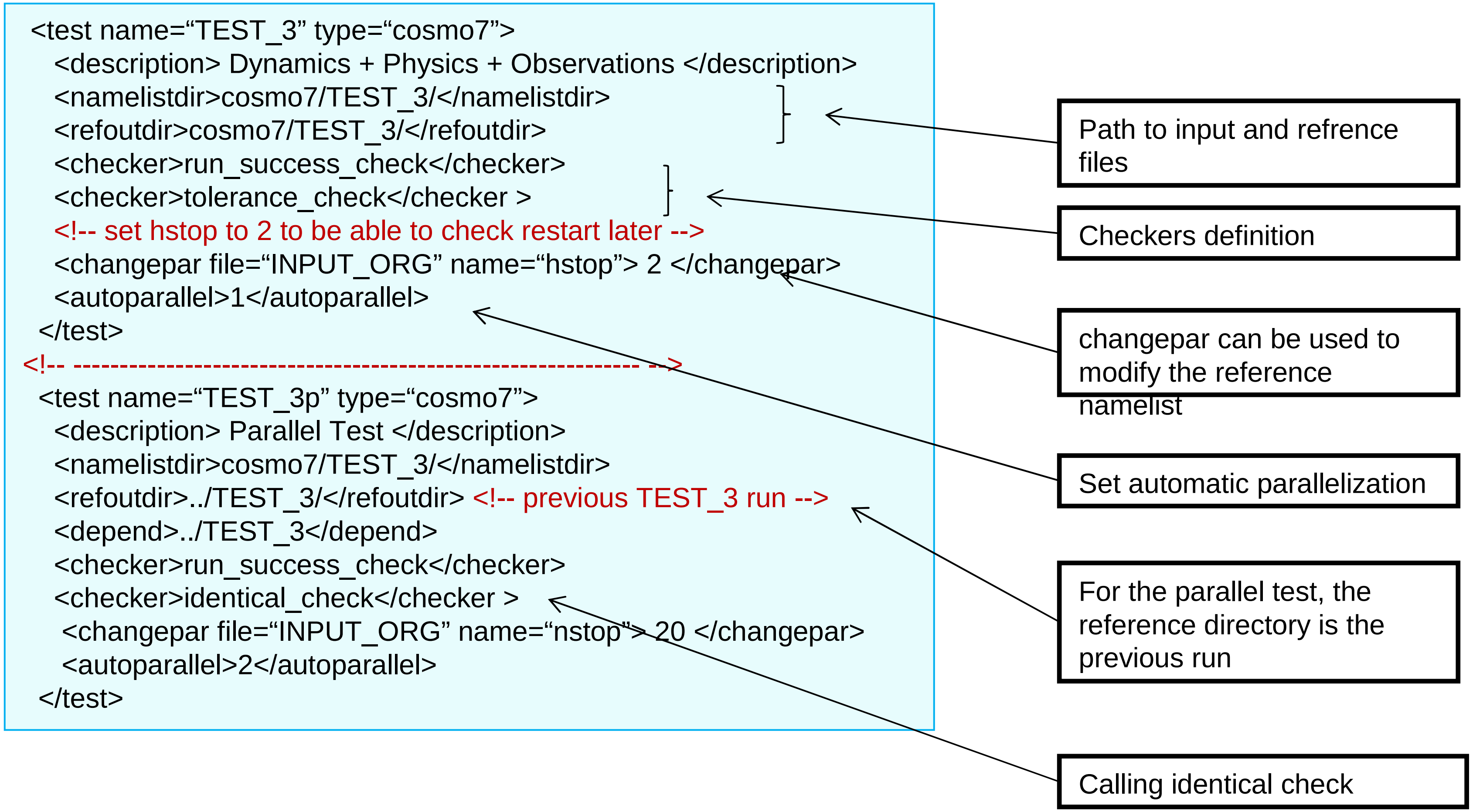
Test_2/ : reference2 INPUT_ and YU*

...

For each test a separate working directory is created ex: Work/cosmo7/TEST_1

Test definition

- The tests are defined in an xml file: testlist.xml



```
 <test name=“TEST_3” type=“cosmo7”>
   <description> Dynamics + Physics + Observations </description>
   <namelistdir>cosmo7/TEST_3/</namelistdir>
   <refoutdir>cosmo7/TEST_3/</refoutdir>
   <checker>run_success_check</checker>
   <checker>tolerance_check</checker >
   <!-- set hstop to 2 to be able to check restart later -->
   <changepar file=“INPUT_ORG” name=“hstop”> 2 </changepar>
   <autoparallel>1</autoparallel>
  </test>
<!-- ------------------------------------------------------------- -->
  <test name=“TEST_3p” type=“cosmo7”>
   <description> Parallel Test </description>
   <namelistdir>cosmo7/TEST_3/</namelistdir>
   <refoutdir>../TEST_3/</refoutdir> <!-- previous TEST_3 run -->
   <depend>../TEST_3</depend>
   <checker>run_success_check</checker>
   <checker>identical_check</checker >
    <changepar file=“INPUT_ORG” name=“nstop”> 20 </changepar>
    <autoparallel>2</autoparallel>
  </test>
```

Path to input and refrence files

Checkers definition

changepar can be used to modify the reference namelist

Set automatic parallelization

For the parallel test, the reference directory is the previous run

Calling identical check

Status with respect to COSMO coding document: *6.5 Standard Test Suite*

….

All versions have to pass a standard test suite, which checks some technical issues. The idea is to define such a test suite, that can easily be run at every center. Issues to be checked are for example:

- Portability (not in testsuite.py)
- Independence of processor configurations (MPI and OpenMP) (ok)
- Reproducibility of results with older versions (ok)
- Restart functionality (ok)
- I/O with Grib/NetCDF (possible)
- Tests with array bound checking (not in testsuite.py, user responsibility)
- Possibility to run with input data from different models (GME, IFS, ERA, etc.) (ok, needs reference input files)
- Timings / efficiency (possible, but difficult to get a portable solution)

Open questions

- A set of tests covering the various options used by the different COSMO members should be defined
- In order to run a fast test, we are using reduced domain size (typically 80x60 grid point), is this ok for all tests/purposes ?
- Do we need to have binary inputs for all grid resolutions (2km, 5km, 7km, …) ?
- Support for int2lm (currently not available) ?
- Where should the reference binary inputs, namelists, and reference YUPRTEST files should be stored (so that they can be shared among COSMO members) ?
- How will this be distributed ? With the code ? In a public repository? On the COSMO webpage?
- Who will do the maintenance, support and further development of this code?
- Shall we add a perturbed field option in COSMO (to set tolerance)
- Is there some urgent additional checkers required ? who will implement them ?

Further notes

- The testsuite was used on IBM (ECMWF), MacOSX, Cray, ECMWF, …
- For NEC this would require to install python on the nodes (the testsuite is currently executed from the compute node)
- The testsuite.py was used by Burkhard Rockel for COSMO-CLM. He ask for a NETCDF checker

Time line

- Consolidation of the current prototype (Until end of September)
- Test and review by WG6 chair (Ulrich Schaettler) and CLM community (Burkhard Rockel) (Until end of November)
- 2nd Consolitation (Until end of December)
- First distribution to all COSMO